

# C1\_W3\_Lab\_3\_custom-layer-activation

April 14, 2021

## 1 Ungraded Lab: Activation in Custom Layers

In this lab, we extend our knowledge of building custom layers by adding an activation parameter. The implementation is pretty straightforward as you'll see below.

### 1.1 Imports

```
[1]: try:
      # %tensorflow_version only exists in Colab.
      %tensorflow_version 2.x
    except Exception:
      pass

    import tensorflow as tf
    from tensorflow.keras.layers import Layer
```

### 1.2 Adding an activation layer

To use the built-in activations in Keras, we can specify an `activation` parameter in the `__init__()` method of our custom layer class. From there, we can initialize it by using the `tf.keras.activations.get()` method. This takes in a string identifier that corresponds to one of the [available activations](#) in Keras. Next, you can now pass in the forward computation to this activation in the `call()` method.

```
[2]: class SimpleDense(Layer):

      # add an activation parameter
      def __init__(self, units=32, activation=None):
          super(SimpleDense, self).__init__()
          self.units = units

          # define the activation to get from the built-in activation layers in
          ↪ Keras
          self.activation = tf.keras.activations.get(activation)
```

```

def build(self, input_shape):
    w_init = tf.random_normal_initializer()
    self.w = tf.Variable(name="kernel",
        initial_value=w_init(shape=(input_shape[-1], self.units),
            dtype='float32'),
            trainable=True)
    b_init = tf.zeros_initializer()
    self.b = tf.Variable(name="bias",
        initial_value=b_init(shape=(self.units,), dtype='float32'),
            trainable=True)
    super().build(input_shape)

def call(self, inputs):
    # pass the computation to the activation layer
    return self.activation(tf.matmul(inputs, self.w) + self.b)

```

We can now pass in an activation parameter to our custom layer. The string identifier is mostly the same as the function name so 'relu' below will get `tf.keras.activations.relu`.

```

[3]: mnist = tf.keras.datasets.mnist

(x_train, y_train), (x_test, y_test) = mnist.load_data()
x_train, x_test = x_train / 255.0, x_test / 255.0

model = tf.keras.models.Sequential([
    tf.keras.layers.Flatten(input_shape=(28, 28)),
    SimpleDense(128, activation='relu'),
    tf.keras.layers.Dropout(0.2),
    tf.keras.layers.Dense(10, activation='softmax')
])

model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])

model.fit(x_train, y_train, epochs=5)
model.evaluate(x_test, y_test)

```

```

Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-
datasets/mnist.npz
11493376/11490434 [=====] - 0s 0us/step
Train on 60000 samples
Epoch 1/5
60000/60000 [=====] - 5s 83us/sample - loss: 0.2907 -
accuracy: 0.9155

```

Epoch 2/5  
60000/60000 [=====] - 5s 77us/sample - loss: 0.1417 -  
accuracy: 0.9579  
Epoch 3/5  
60000/60000 [=====] - 5s 78us/sample - loss: 0.1057 -  
accuracy: 0.9677  
Epoch 4/5  
60000/60000 [=====] - 5s 78us/sample - loss: 0.0867 -  
accuracy: 0.9734  
Epoch 5/5  
60000/60000 [=====] - 5s 77us/sample - loss: 0.0737 -  
accuracy: 0.9766  
10000/10000 [=====] - 0s 30us/sample - loss: 0.0755 -  
accuracy: 0.9762

[3]: [0.07552447271542624, 0.9762]

[ ]: